

P1 1154177

RECEIVED

22 APR 2004

WIPO

PCT

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

April 20, 2004

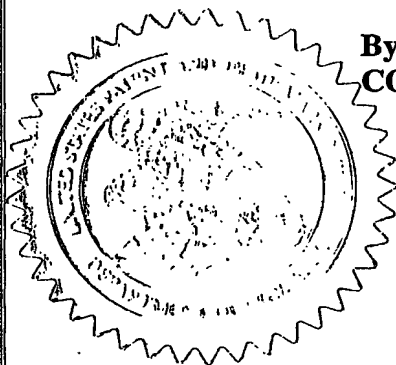
THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE.

APPLICATION NUMBER: 60/442,328

FILING DATE: *January 24, 2003*

RELATED PCT APPLICATION NUMBER: PCT/US04/01458

By Authority of the
COMMISSIONER OF PATENTS AND TRADEMARKS




N. Woodson
N. WOODSON
Certifying Officer

**PRIORITY
DOCUMENT**

SUBMITTED OR TRANSMITTED IN
COMPLIANCE WITH RULE 17.1(a) OR (b)

1133 U.S. PTO
01/24/03

Please type a plus sign (+) inside this box → 

01-27-03 60442328-0121077 *After*

Approved for use through 10/31/2002. OMB 0651-0032
Patent and Trademark Office, U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PROVISIONAL APPLICATION FOR PATENT COVER SHEET

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR 1.53 (c).

Express Mail Label No. EV 139302296 US

INVENTOR(S)		
Given Name (first and middle (if any))	Family Name or Surname	Residence (City and either State or Foreign Country)
Jeremy Mark	Bruestle Tucker	Seattle, WA Seattle, WA
<input type="checkbox"/> Additional inventors are being named on the _____ separately numbered sheets attached hereto		
TITLE OF THE INVENTION (280 characters max)		
METHOD AND APPARATUS FOR SECURE COMMUNICATIONS AND RESOURCE SHARING BETWEEN ANONYMOUS NON-TRUSTING PARTIES WITH NO CENTRAL ADMINISTRATION		
CORRESPONDENCE ADDRESS		
Direct all correspondence to: <input checked="" type="checkbox"/> Customer Number 25096 OR Type Customer Number here		*25096* 25096 <small>PATENT TO ANONYMOUS CORRESPONDENT</small>
<input type="checkbox"/> Firm or Individual Name	Perkins Coie LLP	
ENCLOSED APPLICATION PARTS (check all that apply)		
<input checked="" type="checkbox"/> Specification Number of Pages 61	<input type="checkbox"/> CD(s), Number	
<input type="checkbox"/> Drawing(s) Number of Sheets	<input checked="" type="checkbox"/> Other (specify) Postcard, Check	
<input type="checkbox"/> Application Data Sheet. See 37 CFR 1.76		
METHOD OF PAYMENT OF FILING FEES FOR THIS PROVISIONAL APPLICATION FOR PATENT		
<input checked="" type="checkbox"/> Applicant claims small entity status. See 37 CFR 1.27.		FILING FEE AMOUNT (\$)
<input checked="" type="checkbox"/> A check or money order is enclosed to cover the filing fees		
<input checked="" type="checkbox"/> The Commissioner is hereby authorized to charge filing fees or credit any overpayment to Deposit Account Number: 50-0665		
<input type="checkbox"/> Payment by credit card. Form PTO-2038 is attached.		80.00
The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.		
<input checked="" type="checkbox"/> No.		
<input type="checkbox"/> Yes, the name of the U.S. Government agency and the Government contract number are: _____		

Respectfully submitted,
SIGNATURE 

Date January 24, 2003

TYPED or PRINTED NAME Chun M. Ng

REGISTRATION NO. 36,878
(if appropriate)

TELEPHONE (206) 583-8888

Docket Number: 39125-8002US

USE ONLY FOR FILING A PROVISIONAL APPLICATION FOR PATENT

This collection of information is required by 37 CFR 1.51. The information is used by the public to file (and by the PTO to process) a provisional application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 8 hours to complete, including gathering, preparing, and submitting the complete provisional application to the PTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, Washington, D.C., 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Box Provisional Application, Commissioner for Patents, Washington, D.C. 20231.

JC857 U.S. PTO
60/442328

60442328 . 012403

Express Mail No. EV 139302296 US

COCO PUBLIC TRUST NETWORK MODEL
METHOD AND APPARATUS FOR SECURE COMMUNICATIONS AND
RESOURCE SHARING BETWEEN ANONYMOUS NON-TRUSTING PARTIES
WITH NO CENTRAL ADMINISTRATION.

Jeremy Bruestle and Mark Tucker

CoCo Communications Corp.

January 23, 2003

The following disclosure relates to communications protocols, encryption processes and resource management methods deployed to achieve a decentralized collective network environment where non-trusting nodes can share resources and communicate with an expected quality of service (QOS).

Today's communications requirements have outpaced the capabilities of current systems. There is an urgent need for better security in both voice and data communications and an increased need to protect the transmission stream. These heightened requirements coupled with the need for improved availability and reliability have brought network interoperability and unification to the forefront.

To date, other communication and network models have utilized patches at the application and operating system (OS) level to solve any one of these problems. The security patches alone have spawned an entire industry providing firewall, VPN (Virtual Private Network) and IDS (Intrusion Detection System) enhancements to protocol limitations. The demand for improved communications and the limitations that plague the current models have spawned a need for a new unifying networking model with fundamental changes in structure and architecture. These changes will address the interdependency of security, interoperability, mobility and resource management (including priority and QOS) at the protocol level.

Advancements in radio technology, such as phased array antennas and consumer and enterprise adoption of wireless 802.11 access points, have made it possible to deploy radio mesh networks. Decentralized public mesh networking requires the same changes to the existing network models as those addressed in above. In addition, decentralized public mesh networking requires a new mechanism that allows non-trusting parties to share and control network resources without the need for central administration.

Further more, decentralized public mesh networking requires dynamic grouping for scaling, intelligent optimal route selection for routing within a structure that can support the aggregate inheritance of permissions, and resource allocation to reduce the network control overhead to optimize performance (reduce latency, reduce processor requirements and increase throughput and guarantee QOS).

Part1 Overview

Chapter 1

Overview

The CoCo network is structured as a hierarchical mesh network, with dynamically generated routing tables. Every device on the network is capable of being both an endpoint and a forwarder of communications. The term 'node' is used to refer to an

active component of the network. Node's may differ in capability, but all must be able to follow the basic CoCo routing protocols. Node are connected via any number of underlying network protocols. All underlying networks are represented with one of two models, the link model or the star model. There is a hierarchical network concept which allows groups of nodes to be viewed as a single group, allowing the dynamic routing mechanisms to scale to any size.

End to end connections between non-directly connected nodes are created via a circuit based mechanism similar to asynchronous transfer mode (ATM). One of the two nodes wishing to communicate sends a circuit establishment request. This request is passed through the network based on the routing tables and the requirements of the circuit, and builds the circuit as it goes. Once a circuit is established, all packets traveling through the circuit take the same path.

This circuit mechanism gives the nodes some awareness of the higher level end-to-end connections. This allows quality of service (QOS) levels to be attached to given data flows. Decisions about QOS policy can be made at circuit construction time. These can then be enforced during course of the circuit's existence. In addition, routing is static per circuit, thus once a circuit is created, packet routing is simple and efficient.

The information needed to set up circuits, including inter-node connectivity and link quality of service data is transferred through the network via a hierarchical dynamic routing protocol. Thus each node must be aware only of it's name and it's local connectivity. This information is propagated through the network in a way which reduces the amount of data which needs to be maintained in each node, while providing enough information for reasonably good circuit construction routing choices.

Cryptographic methods are used throughout the protocol to insure the safety, authenticity and correctness of the control data moving between nodes. This allows the end-to-end user data transport to be secure, the dynamic routing to be tamper resistant, the QOS to be cryptographically controlled, and generally prevents nearly all attacks on the network, including many denial of service (DOS) attacks. The assumptions of trust are very pessimistic, while still allowing useful work to be done among many untrusted nodes.

Another important concept of the CoCo network model is that of the "document". A document is a self contained, cryptographically signed, chunk of data understood and used by the network to make decisions. Much of the high level protocols, such as circuit establishment and dynamic routing involve the exchange of documents. The public key distribution mechanism operates on documents.

In addition, documents are used to define policy. This includes user and groups, as well as resource use policies, and delegation of power. Policy documents have names, and one can refer to a policy by its name. Many nodes can share the same policy, and track policy updates.

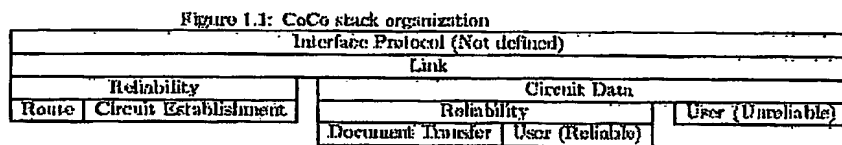
The use of documents requires some mechanism to organize and distribute documents throughout the network, and indeed there are protocols within the CoCo network model to do just this. In addition, some documents, such as routing updates have additional ways of moving themselves through the network that do not rely on the document transfer protocol.

1.1 Protocol Stack

The protocol stack of the CoCo network model does not fit perfectly into the OSI seven layer network model. This is due to the intentional interdependence between the high level end-to-end connection concept and the hop-by-hop routing choices. This allows detailed QOS to be enforced. In addition, security is embedded at a lower layer, again in an attempt to gain further functionality and integration.

The basic protocol stack is shown in Figure 1.1

Figure 1.1: CoCo stack organization



Interface Protocol The underlying protocol used for inter-node communications is referred to as the interface protocol, and is not defined by this specification, although it's requirements are. This protocol need not be consistent throughout the network, indeed it may differ on a node-by-node or link-by-link basis, and there may even be multiple different interface protocols connecting the same nodes.

CoCo Link Protocol Immediately above the interface protocol is the link protocol. This protocol is in charge of the process of initiating CoCo communications with a neighbor, determining the neighbors name, establishing basic link cryptographic communications, etc. In addition, this protocol is the base for CoCo's QOS mechanisms.

CoCo Reliability Protocol There is a lightweight reliability protocol that allows for reliable sequenced streams over unreliable packet traffic. This protocol is used by both the system level and the user level protocols, and is used in multiple locations within the protocol stack.

CoCo Routing Protocol The movement of routing data through the network is known as the routing protocol. This protocol is in charge of making sure the network keeps aware of the current interconnectivity status, as well as the current QOS situation. It deals with the network as a hierarchical system of nodes, networks, networks of networks, etc.

CoCo Circuit Protocol The establishment of end to end connections, and the long haul transfer of data is handled by the CoCo circuit protocol. This protocol is divided into the circuit establishment protocol, which sets up circuits, and the circuit data protocol, which moves data over circuits. All user data is transported over the circuit data protocol, either as raw packets (unreliable circuit), or as reliable streams by use of the reliability protocol (reliable circuit).

CoCo Document Transfer Protocol The CoCo protocol set makes use of user signed data components known as "documents" as described later in this document. Sometimes, a node may need to track down a document that it needs for some reason. The process of finding a document by name and moving the document to the requesting node is known as the document transfer protocol.

1.2 Node Concepts

All devices that speak the CoCo protocol are nodes. Both endpoints and intermediate points are nodes. Each node has a single unique network address. This takes the form of a hierarchical name, and this naming is used to help determine the routing structure and the propagation of the dynamic routing data. The name is represented as a dot separated list of identifiers, with the most specific portion of the node name coming first, and the most general coming last.

For example: `machine1.floor2.seattleOffice.ibm`, the highest level part of the name being of a similar level to the autonomous system number of BGP. This naming format, referred to in this document as the dot path naming format is used to name many aspects of the network, including users, groups, networks, documents, policies, or in this case node. In each case the hierarchy has a well-defined meaning.

1.2.1 Inter-Node Connectivity

Each node is able to communicate with one or more other nodes directly. These nodes are referred to as neighbors, or sometimes as direct nodes. The underlying protocol used by neighboring nodes to communicate is unimportant to the CoCo network layer, so long as it follows a few basic requirements. The following guidelines determine what underlying transport protocols are appropriate:

- May be connection oriented or connectionless.
- May be unreliable or reliable
- Must fit either the link or star model described below
- Must be able to packetize data in some way
- Must have a packet MTU (real or virtual) of 512 bytes or more.
- Must be two way, in that if node A can send to node B, node B can send to node A
- May duplicate or reorder packets

Each node must have a method of determining its neighbors, initiating a physical connection for connection oriented underlying protocols, and exchanging packets. Finding the physical/logical neighbors may be done through some discovery protocol, or the list of desired neighbors could be determined by user or other configuration.

The two basic types of node connectivity

Edge The word edge comes from the graph theory term. An edge is a connection between nodes that represents a simple point-to-point connection. This could be for example a node within radio proximity, or a node on the other end of a private T1. The fact that a node A has edges to two different nodes, B and C has no bearing on

whether or not B and C have an edge connecting them. The requirements of an edge do include bidirectionality and some form of packet based transport, although in the case of a non-packetized physical link, packets could be easily be introduced via a framing mechanism of some sort.

Stars A star is used to represent some form of shared transport mechanism that connects a number of nodes in such a way that the transitive property of connectivity holds among them. An example of a star would be a number of nodes connected to an Ethernet switch. If node A can send packets to node B as well as node C via the Ethernet switch, then both B and C can send packets back and forth via the same Ethernet switch. Thus a star can be thought of as the center of a group of communicating nodes. Note, radio networks are usually NOT stars. Another way to look at stars is as the equivalency classed of connectivity.

1.2.2 Hierarchical Structure

The naming of nodes introduces a hierarchical structure to the network, in that networks of nodes combine to form networks of networks, and networks of networks of networks, etc. A group of nodes, which are part of the same network group, or a group of networks, which are grouped together, are often referred to as a meta-node. This is because whole networks of nodes can be looked at as a single node, and the inter-network connectivity as edges between these meta-nodes.

The highest level meta-nodes form the global network, or root level network. In the Internet model, these would be the networks assigned ASN's. Each of these root level networks, or meta-nodes, can contain internal structure. The depth of the structure before one hits the actual physical node does not have to be consistent throughout the networks. That is, a meta-node composed of three smaller nodes, and a single individual node can be peers in a network. In terms of dot path node names, $a.b.c$ and $x.c$ could both be nodes, and $b.c$ (a meta-node) and $x.c$ (a real node) would be peers in network c .

One requirement for grouping nodes into a meta-node is that all the nodes in the group should be fully connected. That is, for any two chosen nodes in the group, there must be a path consisting of only other nodes within the group, which connects them. Obviously nodes which are on the same star make a great grouping. In some cases, a meta-node may "split" in that the failure of certain links causes the meta-node's components to no longer be fully connected. The CoCo network attempts to deal with this situation, but certain failure modes are possible, and one should avoid grouping nodes in such a way that one or two key links can split the group.

1.2.3 Dynamic Routes

The dynamic routing protocol is in charge of propagating node connectivity and QOS data through the network. The hierarchical nature of the network and the fully connectivity assumption are used to limit the flooding of route messages, and minimize the size of the route tables generated, thus making certain that the dynamic routing scales.

2019年12月31日

1.3.1 Documents in General

Documents are often signed. This signing uses a public key signature mechanism. Interestingly, documents are also used to distribute keys. There is a chain of trust to allow this mechanism to work, as well as a well-known root public key. By signing documents, one can be sure the signer “supports” the document. What “supports” means depends upon the document in question. If the document is an identity document for example, it means the signer vouches that the identity in question is legitimate. If the document is a circuit request, it means the signer is the user requesting the circuit.

All documents have a type associated with them, and many have a name associated with them as well. The type is one of a predefined list of types, and each type has a different internal format. New document types may be added in future versions of the protocol. Unknown document types should be ignored when present. Documents may also have a name. This name takes the form of a "dot path". This is a dot separated list of identifiers, for example `this.is.a.test`. Given a type and a dot path, there is usually a single unique document, and the document transfer protocol is designed to find that document. However not all documents are long-term documents, for example, circuit request or dynamic routing updates. These documents have no names and propagate through some other means, and are called short term documents.

All documents also have a time scope. This may include a creation time, and certainly includes an expiration time. This is the time after which the document should no longer be considered relevant. For example, dynamic route information from a mobile radio device should probably have a fairly quick expiration, as the time in which it is useful is limited. One thing to note is the assumption that there is fairly universal time synchronization among nodes. However, it is assumed that while data

from a given node can be time ordered with great reliability, inter-node times may be off by a few seconds to a minute. One hour time skews are considered unacceptable.

1.3.2 Document Types

There are a number of different document types, and document type classes. Some of these, such as identity documents, are used in conjunction with other documents, in that many other documents require identity documents to be useful. In addition, there are a number of document types involved with the document transfer protocol, or in other words, documents relating to document movement and storage. A quick overview of document types is given here, more detailed descriptions of the document format for a given document type are given in the chapter relating to the use of the document, with a few very basic documents in the chapter 4 on basic document types.

1.3.2.1 Identity Documents

To allow for a very flexible authorization mechanism, there needs to be a way to identify users, nodes, and other logical units of activity. In addition, it is valuable to be able to group these entities to allow for a single rule to be applied to an entire class of entities. To this end, a hierarchical identity naming mechanism is defined. Note, a single human user may have more than one identity for the multiple roles they have. For example, they may have one identity as an employee of their company, and another as a member of a social group. In addition, even within a company they may have multiple roles for the different departments, etc, of which they are a member. Also, non-human components of the network also have identities, for example every node has an identity, which is actually its network address.

Each identity is named via a dot path. Both the 'a.b.c' and 'b.c' can be valid identities at the same time, 'b.c' representing a group, and 'a.b.c' representing an individual or a sub-group. Each identity has a private and public key associated with it. The private key should stay with the human or piece of equipment, which the identity represents, and the public key is distributed.

The identity document is a document connecting the identity and its public key. The identity name is the document name for the purposes of the document transfer protocol. Each identity document must be signed by the public key of an identity higher in the hierarchy. Thus, a group can choose its users and subgroups, which can in turn choose sub-groups of their own. In addition there is a flag associated with the identity document as to whether the identity is a leaf identity, that is, one which cannot have sub-identities. The root identity document that is the public key needed to sign the c of a . b . c is assumed to exist on all nodes.

The details of this document are given in chapter 4

1.3.2.2 Storage Location Documents

The document transfer protocol is used to find documents by name on the network. In order for it to work, there must be documents, which link names to actual nodes. To this end, there are storage location documents. A storage location document gives

that node names of nodes which contain information about a given sub-tree of document names. For example, there may be a storage location for `ibm.corp.us` which points to a node that is one of IBM's document storage servers.

The name of the storage location document is the tree for which it is authoritative. The node the document points to should either contain every document in that tree, or contain storage location records for more specific sub-tree's for documents it does not store. There may be more than one storage location record for the same name, allowing for multiple document storage servers. For a document storage location to be valid, it must be signed by an identity, which is equal to or higher than the sub-tree it stores.

The details of this document are given in chapter 11

1.3.2.3 Logical Name Documents

Logical name documents can be used to give other types of documents a more human acceptable, or organizationally structured view of the world. Logical name documents can also be used to implement redirection strategies. A logical name document is a logical dot path (it's key for lookup), and an actual dot path and document type. It must be signed by an identity equal or higher than it is logical dot path. It redirects to its actual dot path.

The details of this document are given in chapter 11

1.3.2.4 Route Update Documents

There are a number of routing related documents passed between nodes during the process of moving routing data. All these documents are unnamed and do cannot be lookup up via the document transfer protocol.

The details of these documents are given in chapter 9

1.3.2.5 Metric Policy Documents

Metric policy documents give nodes information to use in deciding the permissions and delegations used to make circuit establishment and data movement decisions. Basically, they store the QOS policies. They each have a dot path name, which allows policies to reference other policy, and to lookup new policy documents as old policy expiration dates expire. The document should be signed by an identity equal to or higher than their name.

The details of this document are given in chapter 10

1.3.2.6 Circuit Request Documents

These are unnamed documents (in that they do not have a dot path and are not stored in a document storage server). They contain actual requests for circuit construction. They should be signed by the identity desiring the circuit to be constructed.

The details of this document are given in chapter 10

1.3.2.7 Link Setup Documents

These are unnamed documents (in that they do not have a dot path and are not stored in a document storage server). They are used to set up a link and exchange keying information used to secure the link. They should be signed by the node initiating the link setup.

The details of this document are given in chapter 8

Part2

Detailed Description

Chapter 2

Conventions and basic types

In this chapter, certain conventions used to describe the protocols in this document are described. In addition, we will introduce some basic fundamental data types which will be used to build more complex ones.

2.1 Conventions

2.1.1 Typing

In order to describe both in memory state as well as the logical form of various data moved across the network, a simple, semi-formal typing system is introduced. Given parts of data are represented via various types and there are a number of ways of combining simple types into more complex types. The details of in-memory representation used in actual implementation are not important, and indeed the types used do not need to be the same, as long as the semantics are preserved.

2.1.2 Encoding

In addition to the logical form of types, some types have an encoding as well. This encoding represents the way the type is turned into a sequence of bytes for movement over the network. The encoding should match the description identically, in order for multiple implementations of the protocol to interoperate.

2.1.3 Protocol Data Units

A protocol data unit (or PDU) is a named type which is also a basic conversational unit of the protocol. At the lowest level, the protocol can be looked at as the exchange of PDUs, and the associated change in state of the various nodes involved. In addition to these physical PDUs there are also cases where a logical PDU description is used to represent an application programmer interface (API) to lower level aspects of the implementation. A PDU like description is also used to represent the high level API that the CoCo protocols export. Thus, interfacing with other components of the system is looked at as similar to sending and receiving data with

other nodes. The actual implementation need not use this metaphor, but the descriptions all do.

2.2 Basic Types

2.2.1 TYPE: Integer types U# and S#

Description

The integer type are U8,U16,U32,U64,S8,S16,S32,S64 and represent unsigned and signed integers of various size ranges. The U or S represents unsigned or signed respectively and the number following the letter represents the number of bits used to represent the number. Thus, a U8 can store a number from 0 to 255, and a S16 can store a number from -32768 to 32767. Note: in general, given the number of bits n , unsigned numbers go from 0 to 2^n-1 , and signed numbers go from -2^{n-1} to $2^{n-1}-1$.

Encoding

The encoding of the integer types as byte sequences is basic two's complement binary representation. For multi-byte integers, network byte order (big-endian) representation is used.

2.2.2 TYPE: VarInteger

Description

The variable integer type is used to represent unsigned integers. Logically, it is capable of storing a number between 0 and 2^N-1 . The variable nature comes in it's encoding, where N is a self-sizing variable size, with smaller numbers taking up less space.

Encoding

If the value of the number is between 0 and 127 (2^7-1), the encoding of the value is one byte, standard binary integer representation. If the value is between 128 and $2^{14}-1$, it is stored in two bytes. This is a standard network order two byte integer, with the exception that the most significant bit of the first byte is set to 1. If the value is greater than $2^{14}-1$ the number is stored as a standard network order four byte integer, with the exception that the two highest order bits of the first byte are both set to 1.

2.2.3 TYPE: TimeLocation

Description

The time location type represents a point in time. That is, a particular moment. For example, Jan 2, 2002, 7:30 PM, CST could be one way to represent a time location. Time locations represent actual absolute time, not wall time, and are thus not effected by time zones.

Encoding

Time locations are encoded as an S64 representing the number of nanoseconds since Jan 1, 2001, UTC.

2.2.4 TYPE: TimeDuration**Description**

The time duration type defines a relative difference between two time locations, or in other words, a duration.

Encoding

Time duration is represented as an S64 representing the number of nanoseconds that make up the duration.

2.3 Cryptographic Types**2.3.1 TYPE: Hash****Description**

The hash type represents a cryptographic hash of a byte sequence. When the documentation refers to the hash of a logical structure, it means the hash of the byte sequence which is the encoding of the logical structure described. When a hash is referred to as being over a number of logical components, it is assumed to be the hash of the byte sequence formed by concatenating the encoding of each component in the order they are mentioned in the description of what to be hashed.

In addition a hash may be "keyed" by some value. This means that the key is encoded and placed both before and after the encoding of the object being hashed, and the hash value is the result of hashing this new sequence. This keying allows for symmetric key signatures, in that one can verify the sender of a piece of data hashed in a keyed manor knew the same hash key.

Encoding

While future implementations of the protocol will support multiple, negotiable cryptographic hash algorithm choices, the current version uses the MD5 cryptographic hash. The details of turning a sequence of bytes into the hash, and representing the resulting 128 bit hash as a sequence of 16 bytes is described in a document available from RSA.

2.3.2 TYPE: Signature

Description

The signature type represents a public key signature of some data. Generally, due to the constraints on the size of the data the signature is made over, this signature is of a cryptographic hash, which is a hash of some section of data. So long as the cryptographic hash is trusted, the public key signature can be thought of as being over the entire data block which is the input to the hash.

Encoding

While future implementations of the protocol will support multiple, negotiable public key signature algorithm choices, the current version used the RSA public key signature algorithm. The details of this algorithm and the representation of the resulting signature are described in a RSA document.

2.3.3 TYPE: PublicKey

Description

The public key type represents a public key capable of being used for encoding and signature verification with a certain public key encryption and signing mechanism. If the encryption and signing do not use the same key, a public key type will contain both keys.

Encoding

While future implementations of the protocol will support multiple, negotiable public key signature and encryption algorithm choices, the current version used the RSA public key algorithms. The key is a 2048 bit RSA key, encoded as described in the appropriate prior art RSA documentation.

2.3.4 TYPE: SymmetricKey

Description

The symmetric key represents a key which can be used to do encryption and decryption using a symmetric cipher.

Encoding

While future implementations of the protocol will support multiple, negotiable symmetric key ciphers, the current version uses DES, the key format of which is developed and documented by RSA.

2.3.5 TYPE: PublicEncoded

Description

The public encoded type represent a public key encoding of some data. Generally, due to the constraints on the size of the data the encoding is made over, this encoding is a symmetric key, which is used to encode other data

Encoding

While future implementations of the protocol will support multiple, negotiable public key encoding algorithm choices, the current version used the RSA public key encoding algorithm. The details of this algorithm and the representation of the resulting encoding are described in RSA documentation.

2.3.6 TYPE: PreSymmetric

Description

This type represents the initialization vector or other preamble needed to make the symmetric cipher secure against certain attacks.

Encoding

While future implementations of the protocol will support multiple, negotiable symmetric key ciphers the current version uses DES, the IV format of which is described in certain RSA documentation.

2.4 Combining Types

In order to build complex type from simpler types, there must be a way of combining or modifying types. This section describes the mechanisms used to do just that. There are really three primary complex types, Sequence, Set, and Structure types. All of these types have a well-defined method of encoding, which is related to the method of encoding its base type or types.

2.4.1 Sequence

A sequence is an ordered list of elements all of the same type. It is represented by Sequence<Type> where Type is the type of all the sequence elements. Typical

operation on a sequence might include inserting and removing elements, finding an element based on its location, iterating through the elements in order, etc. When describing an offset within a sequence, a zero based index is used.

The in memory representation of a sequence is implementation dependent and may vary from one context to another, even for sequences of the same type, based on how they are manipulated.

The encoding of a sequence as a sequence of bytes is the concatenation of the number of elements, represented as a `VarInteger`, with the encoding of each of its elements in order.

2.4.2 Set

A set is an unordered collection of elements all of the same type. It is represented by `Set<Type>` where `Type` is the type of all the set elements. Typical operations on a set might include finding elements which meet a certain requirement, set unions and intersections, modification of certain elements (which is equivalent to removing the original element and inserting the modified one). Another operation might be the transversal of the set in a random order, or some specific order based on a comparison function on the elements. Note, like mathematical sets, there cannot be two "copies" of identical elements within the set. All the uses of sets described within this document avoid this case.

The in memory representation of sets is often quite complex. While the set may be simply a collection of elements, the need to find elements based on some property quickly often necessitates some form of indexing. Indeed, in this document, sets are often used as maps, which relate one element with another, by inserting the key-value pairs into the set. In addition, the need to transverse the set in some particular order often induces some complexity. However as this is an implementation issue, it will not be discussed further.

The encoding of a set as a sequence of bytes is the concatenation of the number of elements, represented as a `VarInteger`, with the encoding of every element of the set, in no particular order.

2.4.3 Structure

A structure consists of a sequential, named list of elements, each of its own type. It is used to combine various different parts to form a single whole. A typical definition looks like:

Structure

<code>TypeOne</code>	<code>NameOne</code>
<code>TypeTwo</code>	<code>NameTwo</code>

In this case, we have two parts, of types `TypeOne` and `TypeTwo` respectively. The names in the case of real structures help explain the meaning of the structure and give a way to refer to elements.

The encoding of a structure is simply the concatenation of the encoding of all the parts of the structure in the order given.

Chapter 3

Document Encoding

Document encapsulation is the name for the process of converting the logical form of a document into a sequence of bytes for transfer via some network protocol, either the reliability protocol, or in the case of the link protocol, a special fragmentation mechanism. This chapter describes the relation between the logical and “physical” form (as the octet stream is referred) of the document.

3.1 Component Types

First we introduce some common, sub-components of documents and describe how to turn them into sequences of bytes

3.1.1 TYPE: DotPath

Description

The dot path is a dot separated list of identifiers. It is used in multiple locations within the protocols. Each identifier can be any textual sequence of printable glyphs, so long as it does not contain a “.” which is the separating character. The sequence must begin and end with an identifier, and the dot can only appear in the interior. Thus “a.b.c.” is illegal as there is a dot at the end with no identifier that follows it.

Encoding

The dot path is encoded as a VarInteger representing the number of bytes in its UTF-8 representation, followed by the unicode UTF-8 of the actual dot path in a textual manner.

3.1.2 TYPE: DocumentType

Description

The document type is an enumeration type. Its definition is as follows:

Name	Number
Identity	?

StorageLocation	?
LogicalName	?
ControllerFlood	?
CosignRequest	?
NodeFlood	?
EdgeFlood	?
NextHopFlood	?
InterconnectionFlood	?
MetricPolicy	?
CircuitRequest	?
LinkSetup	?

Encoding

The document type is encoded as a VarInteger representing the enumeration number.

3.1.3 TYPE: SignatureBlock

Description

The signature block represents the signature of some thing, usually a cryptographic hash, by a particular identity. Its structure is as follows:

Structure

DotPath	SignerIdent
Signature	SignerSig

where SignerIdent is the identity that is signing and SignerSig is the cryptographic signature of that particular signer.

Encoding

Standard structure encoding

3.2 Basic Document Structure

3.2.1 TYPE: Document

Description

The document type represents the general format of a document, and the associated encoding. A portion of the document, referred to as the document body, is the type dependent portion of the document.

Structure

DocumentType	Type
DotPath	Name
TimeLocation	Expiration
ByteSequence	DocumentBody
Hash	DocumentHash
Set<SignatureBlock>	Signatures

where:

Type is the type of the document, an element of the enumeration given in earlier.

Name is the name of this document. For example, identity documents which contain the public key for a given identity are named with the identity they define. This is also the lookup key for the document transfer protocol. For unnamed documents types, this field is omitted from the structure.

Expiration is the time after which this document is no longer valid

DocumentBody is the main matter of the document, and it's meaning depends upon the Type given earlier. It is assumed that each document type describes a body encoding, and that this encoding is self sizing, thus the main document decoding logic will be aware when the document body is done.

DocumentHash is the cryptographic hash of the elements Type,Name (if present) Expiration, and DocumentBody in that order.

Signatures is a list of signatures to this document by various parties. Each signature is a public key signing of the hash in DocumentHash.

Encoding

Standard structure encoding, with the exception that Name is omitted for unnamed document types.

Chapter 4

Basic Document Types

This chapter describes the basic document type which are used by multiple subsystems. Currently, there is only one such document type: the Identity document.

4.1 Identity Document

The identity document is used to connect an identity name with the identities public key. An identity document must be signed by an identity higher on the signing tree to be valid. That is, the dot path of the signing identity must be proper suffix of the dot path of the identity document being signed. For example a.b.c could be signed by b.c but not d.c or x.a.b.c, etc.

4.1.1 TYPE: IdentityDocumentBody

Description

The body of the identity portion is simply as `PublicKey`, which is the public key of the identity named in the document name.

Encoding

Standard encoding for the `PublicKey` type.

Chapter 5

Link and Circuit Metric Parameters

The link and routing protocols collect data for the purpose of determining QOS capabilities along certain paths. Users constructing circuits ask for certain QOS properties. In addition, there may be other factors, which the user might want to affect circuit routing, such as security clearance level, or other similar abstract aspects.

To allow for a general mechanism to be used for the propagation and manipulation of such data, the concept of a "metric parameter" is introduced. This is an aspect of a circuit or a link which might be a reasonable part of the decision process of circuit routing. All QOS parameters fall into the idea of metric parameters, as well as a number of more abstract things.

Thus each metric parameter represents a single measurable quantity of some sort. For example, bandwidth would be a metric parameter, as would latency. Each metric parameter has it's own units in which it is measured. For example bps, or microseconds for bandwidth or latency respectively.

In addition, there is support for two basic type of metric parameter values, numeric and symbolic. Symbolic parameters represent some logical aspect of the network the user might care about, such as "division" (for a corporation's internal use) or "ISP" (for the internet). These aspects may also effect routing choice.

Finally, as negotiations involve asking for a required and desired amounts for various parameters (for example, I require 64 kbps, but would like 100 kbps), there must be an understanding of which direction is "desirable". In the case of bandwidth, high numbers are desirable, in the case of latency, low numbers. In the case of symbolic parameters, which are given using `DotPath` naming, more specific or less specific may increase desirability.

5.0.1 TYPE: MetricParameter

Description

The metric parameter type is used to specify which parameter is being discussed. To this end, each parameter is given a number, thus this type is an enumeration. Additionally, the metric parameters can be divided into four categories, NumericLow, NumericHigh, SymbolicSpec, SymbolicGen, where the first part names the basic type, and the second part specifies which direction is desirable. The type of numeric types is a VarInteger. The parameter type is a U16, with the first two bits deciding the category. An initial list is:

Table of Metric Parameters

Name	Units	Type	Number
Bandwidth	bps * 100	NumericHigh	0
Latency	microseconds	NumericLow	16384

Encoding

Standard U16 encoding

5.1 Uses

This metric measurement idea is used for a number of uses, these include:

- Measuring total and current available capacity/metrics on a both a per-circuit and cumulative bases
- Specifying resource allocation policies.
- Specifying circuit requirements/desires

The first is used in the routing protocol, the second in policy management, the third in circuit construction. The types for all these are described below

5.1.1 TYPE: CurrentMetricData

This type give the total and current metric data of an edge. Included are the total and current best that a single path could receive, as well as the total and current cumulative over all paths. The structure is as follows:

Structure

MetricParameter	WhichMetric
VarInteger	PerPathBestCapacity
VarInteger	PerPathBestUnused
VarInteger	TotalCapacity

VarInteger

TotalUnused

In the case of symbolic types, the structure is actually:

Structure

MetricParameter

WhichMetric

DotPath

Value

where there is only one allowable value for the edge. This may be changed in future protocol versions.

Encoding

Standard structure encoding, which choice of structure determined by the first element

5.2 Metric Policy

The metric policy documents determine the division of resources into different uses, particularly nodes and users and sub-policies. Each policy has a name, which acts as the key for the policy document via the document transfer protocol. The policy then divides resources into groups and assigns each of the groups to a user, a node or meta-node, or a sub-policy. A given node is assigned a single master policy which is used to divide resources going in and out of it.

5.2.1 TYPE: SubholderType

Description

Sub-holder type is used to define which of the three types of sub-holders is being used. It is a simple enumeration as follows;

Subholder Enum

Name	Number
Sub-Policy	?
User	?
Node	?

Encoding

Standard U8

5.2.2 TYPE: MetricPolicyComponent

Description

Policy only applies to numeric metric parameters. This type represents the information about the absolute and relative access to the resource defined by a given metric parameter is and part of the policy document typing. Its structure is as follows:

Structure

MetricParameter	ParameterType
VarInteger	ReservedAmount
VarInteger	AbsolutePriority
VarInteger	RelativeOverflowPriority

Encoding

Standard structure encoding.

5.2.3 TYPE: MetricPolicyPart

Description

The policy part represents one of the eventual sub-holders of the resources granted to a policy. It describes the absolute and relative privileges of the sub-holder. It's structure is as follows:

Structure

SubholderType	Type
DotPath	SubholderName
Set<MetricPolicyComponent>	Params

Encoding

Standard structure encoding.

5.2.4 TYPE: MetricPolicyPart

Description

This is the body of the metric policy document. The key to a metric policy document is the name of the policy. The policy must be signed by an identity more general than the policy name. The structure of a policy is as follows:

Structure

Set<MetricPolicyPart	Subparts
----------------------	----------

Encoding

Standard structure encoding.

5.3 Metric Requirements

The third and final use of the metric system is specifying requirements during circuit construction. This is done using the following types:

5.3.1 TYPE: SingleRequirement

Numeric Case:

Structure

MetricParameter	Parameter
VarInteger	Required
VarInteger	Desired

Symbolic Case:

Structure

MetricParameter	Parameter
DotPath	Required
DotPath	Desired

Encoding

Standard structure encoding, choice of structure based on first element

5.3.2 TYPE: MetricRequirements

TDesc This type defines the requirements of a circuit establishment

Structure

Set<SingleRequirement>

Requirements

Encoding

Standard structure encoding.

Chapter 6

Interface API Requirements

6.1 Introduction

The node calls a method on the interface API, which then uses the underlying interface's native protocol to transmit a packet, or perform some requested operation. Packets and events that enter the interface via the native protocols are converted to callbacks in the interface protocol and sent up the CoCo protocol stack. Thus the interface API is the layer of abstraction that allows many types of physical networks to be dealt with in an equivalent manner. The process of calling API's and receiving callbacks is very similar to sending and receiving packets.

It is assumed this underlying protocol may have some addressing mechanism to allow links to multiple nodes to be reached via the same physical/logical interface, but it may be a simple point-to-point link as well. These remote addresses are interface specific and opaque to the higher-level protocols. The underlying protocol may be connection orientated, in which case the addresses define the locations to connect to, otherwise the addresses define a per-packet destination. The underlying protocol must be packet based and each interface must have or be given a well defined MTU for the data portion of the packet.

In addition, some interfaces (such as radio) will wish to alert the CoCo network layer of newly discovered remote nodes. Also, some interfaces (such as Ethernet) will support broadcast of some form and should allow some method of node resolution, similar to ARP. The API supports both of these mechanisms.

6.2 Interface Protocol Base Types

Here the basic data types used for communication with the interface API are introduced.

6.2.1 TYPE: RemoteAddress

Description

An opaque representation of the remote address, must be understood by the interface. This type is never sent over the network, except by the interface itself perhaps, and thus does not have an encoding. Thus any structure containing it does not have an encoding.

6.2.2 TYPE: SeekRequestID

Description

A number used by the higher level protocol to associate requests with responses. The in memory form is identical to a U32. This type is never sent over the network, except by the interface itself perhaps, and thus does not have an encoding. Thus any structure containing it does not have an encoding.

6.3 Read and Write

6.3.1 API: SendPacket

Direction: To Interface

Description

This call sends a packet down to the interface to be transmitted. Its parameters are:

Parameters

RemoteAddress	Address
ByteSequence	PacketData

where `Address` is the remote address to send the packet to, and `PacketData` is the packet data, which must be smaller than the MTU of the interface.

Errors

Bad Remote Address Remote address is of the wrong form, or was not a remote address which is "current". Current remote addresses include those found though seeking or detection, and which have not had an address down event

6.3.2 API: ReceivePacket

Direction: From Interface

Description

This is called by the interface when it receives a packet. It's parameters are:

Parameters

RemoteAddress	Address
ByteSequence	PacketData

where **Address** is the remote address the packet arrived from, and **PacketData** is the packet data, which will be smaller than the MTU of the interface.

6.4 Seeking

6.4.1 API: SeekNode

Direction: To Interface

Description

This call sends a packet down to the interface to be transmitted. It's parameters are:

Parameters

SeekRequestID	RequestNumber
DotName	NodeName
TimeDuration	Timeout

where **RequestNumber** is a U32 used to associate the resulting callbacks with this call, **NodeName** is the node name of the node being sought, and **Timeout** is the amount of time the node seeking is willing to wait to find the node sought.

Errors

Request Number In Use	The request number is already the number of a pending seek
Invalid Node Name	The node name is not structurally valid
Node Not Seekable	The interface knows that the node being sought will not be found, for example, the node name is not local to the physical network

6.4.2 API: SeekFound

Direction: From Interface

Description

This is called by the interface when it finds a node matching an outstanding request. After the seek found, the request ID is no longer in use and can be reused. Its parameters are:

Parameters

SeekRequestID	RequestNumber
RemoteAddress	Address

where RequestNumber is the request this response is in relation to, and Address is the remote address of the found node. This remote address is now current (packets can be sent on it).

6.4.3 API: SeekFailure

Direction: From Interface

Description

This is called by the interface when it times out trying to seek a node on the behalf of a caller. After the seek failure, the request ID is not longer in use and can be reused. Its parameters are:

Parameters

SeekRequestID	RequestNumber
---------------	---------------

where RequestNumber is the request this response is in relation to.

6.5 Detection and Address Down

6.5.1 API: DetectNode

Direction: From Interface

Description

This is called when the interface detects another node on the network. This may happen when a node enters radio range, or for example when another node on a broadcast network sends a seek with the local node name. Its parameters are:

Parameters

RemoteAddress	Address
DotPath	NodeName
Bool	RemoteActive

where `Address` is the address of the node detected, and `NodeName` is the probable name of the node. After receiving this call, `Address` is current (can have packets written to it). `RemoteActive` is a flag which indicates some level of initiative on the part of the remote node, and is a hint to not begin link negotiation, as the remote side intends to.

6.5.2 API: AddressDown

Direction: From Interface

Description

This is called to alert the node that an address is no longer current and that the node should no longer send packets to it. Its parameters are:

Parameters

RemoteAddress	Address
---------------	---------

where `Address` is the address which is no longer current.

Chapter 7

Reliability Protocol

7.1 Introduction

Instead of building TCP stream like reliability into a particular layer, a lightweight reliability protocol is used throughout the design. The protocol needs room for three flag bits, and two 32 bit numbers, a sequence and acknowledge. It turns an unreliable, two way, packet transmission mechanism into a reliable one. The mechanism is the same as that used by TCP. One difference is since the concept of a channel (IP & Port) has been separated from the reliability aspect, there is no reason to require the two directions of the connection to open at the same time, thus one way reliable connections are allowed.

Only one stream in either direction is allowed per lower level two way connection. Attempting to start a new connection before receiving a confirmation of

closing of the previous one is not allowed. The virtual PDU definition for the reliability layer is given below, the actual components may be placed into the underlying packet in any manor desired.

The details of the reliability protocol are not described in this document due to the fact that they are semantically identical to the reliability mechanisms used by TCP.

7.1.1 PDU: ReliableChannel

ReliableBoth

7.2 States

7.2.1 TYPE: RelabilitySendState

Structure

Chapter 8

Link Protocol

8.1 Introduction

The link protocol is the lowest level of communications between nodes. It is the first line of defense, dealing with the safety and management of all inter-node data, as well as being the base construct from which QOS is derived. The goal of the link protocol is are:

- Verify the identity of the remote node
- Prevent the reading and/or modification of data moving over the link
- Prevent denial of service attacks based on CPU overloading, table overflows, and make certain that packet insertion cannot damage the transmission of other unmodified packets
- Define the base QOS measurements used by higher level protocols
- Create a consistent inter-node transmission mechanism.
- Create a reliable sub-channel for higher level protocols to use for control packets

To this end, the link protocol establishes the identity of both sides and exchanges a symmetric key used for encryption and per-packet verification. Because of the

computational complexity of the necessary public key mechanisms used during link establishment, and the danger of a DOS attack based on CPU starving, a node will only spend a portion of its CPU on link establishment. To prevent legitimate attempts from being turned away, a computational challenge is implemented to make the DOS attacker unlikely to be able to send 'correct' requests at a reasonable rate. It is assumed that the correctness testing method is fast enough to be checked without penalty.

In addition, no state is kept for challenge requests, only for successful challenge replies, thus DOS attacks on in memory tables should also be able to be avoided. Also, cryptographic methods are used to prevent false replies, etc.

All of this complexity is used to transfer a single document, known as the Link Setup document, which contains the public encrypted and signed symmetric key. Once this document is processed, the link will be considered established, and a reply will be returned. Then the link will be used to transfer symmetrically encrypted, cryptographically checksummed packets, along with QOS information, which composes the "activated" portion of the link protocol.

8.2 Link Documents

8.2.1 TYPE: LinkSetupBody

Description

The link setup document is used for initial link key exchange. It is an unnamed document. It should be signed by the initiating node. Its structure is a single symmetric key, publicly encoded to the destination node.

Encoding

Standard PublicEncoded encoding.

8.3 Link Document Packetization

8.3.1 TYPE: LinkDocumentPart

Description

The link setup document, combined with the full identifying documentation of the initiating node, all concatenated together from the link setup documentation. This string of bytes is most likely too large to fit within the link MTU, and thus must be broken into parts. These parts are then cryptographically hashed. The initial request contains the hash for each of the parts, allowing the acceptance of the primary request

to clear the way for the part, without allowing an intervening attacker to forge data before the key has been exchanged.

The form of each part is as follows:

Structure

U8	PartNumber
VarInteger	PartOffset
VarInteger	PartSize
ByteSequence	Data

where:

PartNumber is the integer part number of the current part, 0 based index.

PartOffset is the offset in bytes of the data portion of this part relative to the entire link documentation set.

PartSize is the size in bytes of the data portion.

Data is the actual data of the part, which the cryptographic hash of should have been sent with the link initial request as described in a later section.

8.4 Link Request Type

8.4.1 TYPE: LinkRequest

Description

The link request type contains information about the requester and the hashes of the documentation to follow. The request will result in challenge as described in the next section, and the complexity of the challenge will be up to the receiver. The format of the link request type is as follows:

Structure

DotPath	RemoteNode
U8	NumberOfParts
VarInteger	RequestDocSize
Sequence<Hash>	PartHashes

where RemoteNode is the node name of the node requesting, NumberOfParts is the number of actual data parts that form the documentation as describes in the previous section, and RequestDocSize is the total size of all those parts and PartHashes is the list of hashes of each part, in order of the part number.

8.5 Link Challenge Exchange

The purpose of the link challenge mechanism is to make the requester do work before the link will even check the validity of the link documentation, as the storing, reassembly, and cryptographic verification of the link documentation is a non-trivial process and allowing any unverified node to force the checking node into doing such work could result in resource starvation.

To this end, a node may request a 'challenge' from the node it wishes to establish a link with. The node receiving this request makes a new challenge, which is a CPU constrained cryptographic task, and signs the challenge. To continue the protocol, the requesting node must return the signed challenge along with the solution. Note: until it receives the solution, the receiving node does not need to store any state, or perform any large computational functions. Upon receiving an apparent solution, the process of checking the solution is reasonably quick.

The challenge chosen for this version of the protocol is searching for a missing portion of data which results in a particular cryptographic hash. So long as the hash is strong, this will result in a brute force search of the unknown data portion. Thus the receiving node may make the challenge as difficult or simple as is desired by changing the number of unknown bits.

To make certain that the challenge was truly chosen by the receiving node, and to make sure that the request the challenge is related to is the same as the request the challenge was sent in response to, the challenge and the request data are cryptographically hashed with a key known only to the receiving node.

8.5.1 TYPE: LinkChallenge

Description

The link challenge type is used to represent the actual cryptographic challenge. Its structure is as follows:

Structure

Sequence<U8>	PrePart
U8	NumberOfBits
Hash	Result

where PrePart is the first portion of the data used to make the hash Result and NumberOfBits is the number of additional bits in the hashes pre-image. If the number of bits is not evenly divisible by eight, it is assumed that the final byte has zeros in the low order (less significant) bits.

Encoding

Standard structure encoding

8.5.2 TYPE: LinkChallengeAnswer

Description

This type represents the answer to a LinkChallenge. It has a type equivalent to Sequence<U8>, which is remaining bytes which when combined with the PrePart of the LinkChallenge create the correct hash.

Encoding

Standard Sequence<U8> encoding.

8.6 Link Packet Types

There are a number of link packet types. Since the link protocol exists immediately above the interface protocol, all interface packets arriving are assumed to be link protocol packets. The first byte determines the type of link packet. The names and associated numbers of each link packet type are as follows:

Link Packet Type Table

Name	Number
LinkInitialRequest	?
LinkRequestNAK	?
LinkRequestChallenge	?
LinkChallengeResponse	?
LinkInitialACK	?
LinkInitialNAK	?
LinkPartSend	?
LinkPartNAK	?
LinkFinalACK	?
LinkFinalNAK	?
LinkReliablePacket	?
LinkUnreliablePacket	?
LinkReset	?

8.7 Initial Link Setup Protocol

The initial link setup protocol uses the challenge response mechanism to prepare the receiving node to have the link setup documentation transferred to it, while at the same time, forcing the initiating node to do some computational work. It involves the following PDU's

8.7.1 PDU: LinkInitialRequest

Description

Structure

LinkRequest	RequestData
-------------	-------------

Encoding

Standard structure encoding

8.7.2 PDU: LinkRequestNAK

Description

Structure

LinkRequest	RequestData
-------------	-------------

Encoding

Standard structure encoding

8.7.3 PDU: LinkRequestChallenge

Description

Structure

LinkRequest	RequestData
LinkChallenge	Challenge
Hash	ChallengeSignature
Hash	InitializationKeyHash
Hash	FinalizationKeyHash

Encoding

Standard structure encoding

8.7.4 PDU: LinkChallengeResponse

Description

Structure

LinkRequest	RequestData
LinkChallenge	Challenge
Hash	ChallengeSignature
LinkChallengeAnswer	Answer

Encoding

Standard structure encoding

8.7.5 PDU: LinkInitialACK

Description

Structure

LinkRequest	RequestData
Sequence<Byte>	InitializationKey

Encoding

Standard structure encoding

8.7.6 PDU: LinkInitialNAK

Description

Structure

LinkRequest	RequestData
Sequence<Byte>	InitializationKey

Encoding

Standard structure encoding

8.8 Secondary Link Setup Protocol

8.8.1 PDU: LinkPartSend

Description

Structure

LinkDocumentPart Part

Encoding

Standard structure encoding

8.8.2 PDU: LinkPartNAK

Description

Structure

U8 PartNumber

Encoding

Standard structure encoding

8.8.3 PDU: LinkFinalACK

Description

Structure

Sequence<Byte> FinalizationKey

Encoding
Standard structure encoding

8.8.4 PDU: LinkFinalNAK

Description

Structure

Sequence<Byte> FinalizationKey

Encoding
Standard structure encoding

8.9 Link Use Protocol

8.9.1 PDU: LinkUnreliablePacket

Description

Structure

PreSymmetric	IV
U16	Seq
U16	Timestamp
Hash	Hash
Data	Data

Encoding
Standard structure encoding, but encrypted with the link negotiated symmetric key

8.9.2 PDU: LinkReliablePacket

Description

Structure

PreSymmetric	IV
U16	Seq
U16	Timestamp
Hash	Hash
U32	SendSeq
U32	AckSeq
U32	WindowSize
Data	Data

Encoding

Standard structure encoding, but encrypted with the link negotiated symmetric key

8.9.3 PDU: LinkReset**Description****Structure**

PreSymmetric	IV
U16	Seq
U16	Timestamp
Hash	Hash

Encoding

Standard structure encoding

Chapter 9**Routing Protocol****9.1 Introduction**

The CoCo routing protocol is designed to give nodes the information necessary to correctly move circuit establishment requests closer to their final destination. It should also allow nodes to make intelligent choices about the path used based on the metric parameters of the various links and the requirements of the circuit request.

To allow the routing protocol to scale, networks are assumed to be hierarchical in organization. Each node has a node name, and the routing first gets the circuit request to the most general part of the node name, then as the node gets to the most general network, the routing information is more specific, allowing the next portion of the node name to be examined. This process continues until the node arrives at its final destination.

A network of nodes is collectively called a meta-node. If nodes in two different meta-nodes are connected via an edge, then the two meta-nodes are connected via a meta-edge. There is a mechanism to combine multiple links between meta-nodes into a single meta-edge. Edge and meta-edge information is flooded though the network, with certain rules to limit the data to relevant areas. The highest level meta-edge data flood everywhere.

The meta-edge data is generated by special meta-node controller nodes. These nodes must have the meta-node key as well as their node key. There may be more than one meta-node controller, as they should all come to the same result, and flooding rate is controlled via each node, thus additional meta-node controllers will simply send extra updates which will be ignored if they are excessive.

It would be possible to not have meta-node controllers, however, this would require all nodes to contain the meta-node key (bad), or the security of the meta-link data to require only the signature of one node in each meta-node of the meta-link, which is possibly acceptable, but still more open to compromise than the meta-node controller mechanism, since it requires compromise of a controller in each meta-node.

Additionally, the meta-node controller mechanism allows much of the routing computation to be somewhat centralized, while still allowing for full failure handling, since there can in theory be any number of meta-node controllers.

Another point worth noting, all star networks must be related to a meta-node. This is a limitation of the current protocol description, and may be removed in the future, although it may actually be the best design decision, as star networks by nature qualify as meta-nodes. In addition, nodes in star networks never exchange network routing data for the local network, but only communicate with the meta-node controllers to exchange connectivity information.

9.2 Routing State

Before describing the protocol used to moving routing information around, we describe state of nodes, as well as meta-node controllers. A node may be a meta-node controller for any network level, and possibly more than one at a time. If a node is a meta-node controller for a given level it has full routing information for that level. Otherwise, a node has partial routing information for a given level. All nodes have full routing information for the local network segment.

The partial routing information is actually derived from the full routing information. The partial routing information is the only information used to actually route circuit requests. We describe the form of the full routing information first, then the partial routing information, then the algorithm to generate the partial information from the full.

9.2.1 Full Routing State Introduction

The full routing information consists of all the edges or meta-edges that are part of the network, along with the metric parameters of each edge, and all the nodes or meta-node which are part of the network, along with the metric parameters of the node-interconnect. For the purpose of this discussion the words node and edge will be assumed to possibly mean meta-nodes or meta-edges.

9.2.2 TYPE: EdgeInformation

Description

The edge information type has information about a single edge. The format of the information is as follows:

Structure

```
DotPath      NodeA
DotPath      NodeB
Sequence<CurrentMetricData> MetricData
```

where NodeA and NodeB are the two nodes that are connected. At least one of these nodes must be in the network for which the full routing information is being stored. The MetricData contains all the most recent total and current free values for the metrics tracked by the edge.

Encoding

Standard structure encoding

9.2.3 TYPE: NodeInformation

Description

The node information type has information about a single node. The format of the information is as follows:

Structure

```
DotPath      Node
Sequence<CurrentMetricData> MetricData
```

where Node is the node in question. The MetricData describe the internal capacity and current free capacity of the node. This is more relevant for meta-nodes, since the ability to move data from one side of the meta-node to the other may be limited. Capacity is given in terms of the worst inter-link case. That is, is a meta-

node connects to three other meta-nodes, A, B, and C, these numbers are the worst case of the meta-nodes ability to move data from A to B, B to C, and A to C.

Encoding

Standard structure encoding

9.2.4 TYPE: FullRouteTable

Description

The full route table contains full routing data for a given network at a given level. It's format is:

Structure

Set<EdgeInformation>	Edges
Set<NodeInformation>	Nodes

where Edges is the set of all the edge information for all edges in the networks, and Nodes is the set of all node information for all nodes in the network.

Encoding

Standard structure encoding

9.2.5 Partial Routing State Introduction

The partial routing state represent the calculated next-hop routing information specific to a given viewpoint within the network. Given a final destination (to the resolution of the network being routed) it results in a list of the destinations one hop from the current location, along with metric information about the total path metric for each of the next hop possibilities. In addition, in the case of meta-routing data there is another table giving all the neighboring meta-nodes, and the edge nodes one level lower which have a connection to the neighboring meta-nodes and the metric data of that particular lower level edge.

9.2.6 TYPE: NextHopEntry

Description

The next hop entry is a particular possible next hop to reach a certain final destination and the associated full path calculated metric information.

Structure

Node	FinalDestination
Node	NextHop
CurrentMetricData	MetricData

Encoding
Standard structure encoding

9.2.7 TYPE: InterconnectionEntry

Description

The interconnection entry contains the information about a lower level node or meta-node which acts as an edge node between two meta-nodes, as well as the associated metric data of the actual edge which connects the meta-nodes. It's structure is as follows:

Structure

Node	NeighborMetaNode
Node	EdgeNode
CurrentMetricData	MetricData

Encoding
Standard structure encoding

9.2.8 TYPE: PartialRouteTable

Description

The partial route table contains partial routing data for a given network at a given level. It's format is:

Structure

Set<NextHopEntry>	NextHops
Set<InterconnectionEntry>	Interconnects

where NextHops is the set of all the next hop information calculated. It is assumed that the set is indexed by the FinalDestination variable of the NextHopEntry structure. It should be reiterated that there may be multiple entries that lead to the same final destination. Similarly, the Interconnects variable should be indexed by the NeighborMetaNode variable of the InterconnectionEntry type, and again there may be more than one entry.

Encoding

Standard structure encoding

9.2.9 Conversion Algorithm

The conversion algorithm is in charge transforming full routes into partial routes. To do this, a method is needed to combine parallel and serially arranged QOS metrics. Each QOS metric must have a parallel combination algorithm (the total QOS between two points based on the QOS on two parallel routes between the points), as well as a serial combination algorithm (the total QOS between two points based on the QOS from the initial point to an intermediate point, and the QOS from an intermediate point to the final point). For example the parallel algorithm for bandwidth is the sum of the two bandwidths, the serial algorithm is the minimum of the two bandwidths.

The conversion algorithm determines for each final destination node, and for each next hop, all the paths to the final node via the next hop. It uses the QOS combination mechanisms to convert the acyclic graph of QOS into a single QOS value, which is associated with the next hop.

9.3 Data Movement Introduction

There are four basic parts to the routing protocol. The first is the meta-node controller flooding protocol which lets all the nodes of a meta-node know about node-controllers. The second is the double signing protocol. This is the method by which one half of an edge or meta-edge sends a partially signed edge to other side for checking and co-signing. The third is base data flooding protocol, by which the edge, meta-edge, node and meta-node data is flooded to everyone who needs to know. The fourth is the method by which, after using the base data to compute, meta-node controller nodes flood the calculated partial data to the local network.

All the routing packets exchanged by nodes are not really packets per se, but documents which are sent over the reliable portion of the link. As mentioned above, most of these documents (with the exception of the cosign requests) are transferred by flooding. Flooding is the process of broadcasting data to everyone on the network who needs to know.

The flooding mechanism works as follows: Each node upon receiving a document from its neighbors checks to see if the document is even relevant to it. If it is not, it is ignored. Next, the node checks if it has seen the same document recently. This is done through a previous document table. To prevent overflow to the table, if a node receives too many documents from a neighbor within a given time, it occasionally drops packets. Either way, if the node has seen the document previously, it drops it. If it is a new, relevant document, the node checks the signature. If it is incorrect, it lowers the allowed data rate for the neighbor from which it received it, as a neighbor should never send a bad document, and the neighbor may be compromised. If all is well, the node checks which neighbors the document is relevant to, and sends it to all of those to whom it is likely relevant.

9.4 Controller Flood Protocol

9.4.1 TYPE: ControllerFloodBody

Description

The controller alert protocol lets nodes know about meta-node controllers. The relevancy criteria is whether or not the node belongs to the meta-node the controller is a meta-node controller for. The format of the document body is

Structure

DotName	MetaNode
DotName	Controller

where MetaNode is the network name of the meta-node that the controller is controller for, and Controller is the full node name of the controller node. Obviously, Controller must be a more specific dot name of MetaNode. The document must be signed by the meta-node key for the meta-node in question.

Encoding

Standard structure encoding.

A given node should remember at least a few controllers for each level of meta-node which it belongs to. It should always keep the closest controllers and drop the further one based on hop count.

9.5 Cosigning Protocol

9.5.1 TYPE: CosignRequestBody

This protocol is used to get an edge update signed by the other party. It involves the transfer of a document known as the CosignRequest document. This is described as

Structure

EdgeInformation	EdgeInfo
-----------------	----------

where EdgeInfo is the edge information from one node being sent to the other to be consigned. It must be signed by the NodeA parameter of EdgeInfo.

Encoding

Standard structure encoding.

The cosign request documents are directed closer to the network named in the NodeB portion of the EdgeInfo entry. If the request has already entered the network named, then node uses it's information about the controllers to send the document closer to the closest known controller.

9.6 Base Flooding**9.6.1 TYPE: NodeFloodBody****Structure**

NodeInformation

NodeInfo

Must be signed by the node or meta-node described in the NodeInfo, relevant if the name of the node with it's first element removed is a generalization of the current node's node name.

Encoding

Standard structure encoding.

9.6.2 TYPE: EdgeFloodBody**Structure**

EdgeInformation

EdgeInfo

Must be signed by both nodes or meta-nodes described in the EdgeInfo, relevant if either node name with it's first element removed is a generalization of the current node's node name.

Encoding

Standard structure encoding.

9.7 Calculated Flooding

9.7.1 TYPE: NextHopFloodBody

Structure

NextHopEntry NextHop

Must be signed by the meta-node for the network layer it describes, relevant if the signing meta-node is a meta-node of the current node.

Encoding

Standard structure encoding.

9.7.2 TYPE: InterconnectionFloodBody

Structure

InterconnectionEntry Interconnection

Must be signed by the meta-node for the network layer it describes, relevant if the signing meta-node is a meta-node of the current node.

Encoding

Standard structure encoding.

Chapter 10

Circuit Protocols

This chapter defines the protocols used to construct circuits and to move data through circuits. It begins by defining the circuit state tables, which are used by circuit data movement and changed by the circuit establishment process.

10.1 Circuit Routing Tables

The circuit routing tables store the current circuits passing through a node. Each circuit connects a remote node on one interface with a remote node possibly on another interface. Since there may be more than one circuit between any two nodes, there is a CircuitID which is used to distinguish them. This CircuitID is local

to the two nodes, that is, it is used only to separate the multiple circuits between two nodes.

10.1.1 TYPE: CircuitID

Description

An identifier which allows the separation of multiple circuits between the same node

Encoding

A standard U32

The route tables store a relation between the two sides of a circuit, and possibly internal data for QOS accounting and control. The type representing one side of a circuit is:

10.1.2 TYPE: CircuitSide

Structure

CircuitID	ID
Interface	RemoteInterface
RemoteAddress	RemoteNode

where RemoteInterface represents the particular interface this side of the circuit goes out of, and RemoteNode is the interface specific RemoteAddress of the other node. The ID is the CircuitID relative to the inter-node connection.

There is a special interface known as "Local". This is the interface used if one side of the circuit is local.

Encoding

Standard structure encoding.

10.1.3 TYPE: CircuitLink

Description

The circuit link defines a relation between two circuit ends, along with implementation specific data to manage the QOS, or other things.

Structure

CircuitSide	SideA
CircuitSide	SideB

10.3 Establishment Protocol

The establishment protocol is used to setup circuits. It involves the movement of a circuit request document through the network over the reliable link protocol. As the circuit request moves, it establishes the circuit. Upon reaching the other side, data can be sent both ways. If the receiving side does not have any initial data to send, it should send a keep-alive to verify the circuit bidirectionally.

The circuit request document has the format as follows

10.3.1 TYPE: CircuitRequestBody

Description

The circuit request body contains the actual circuit request. It should be signed by the user requesting the circuit be established. The format is as follows:

Structure

DotName	DestinationNode
DotName	Service
MetricRequirements	Requirements
U8	Flags
PublicEncrypted	CircuitKey

where

DestinationNode is the final destination of the circuit

Service is the name of the service desired on the other side (similar to a port number and implementation specific)

Requirements are the QOS and other metric requirements of the circuit.

Flags is the flags for the circuit, currently defined are bit 0 (the lowest bit) which is encryption, and bit 1 (second lowest) which is reliability.

CircuitKey is only part of the structure if the flag for encryption is set. If so, it is the symmetric encryption key, public key encrypted to the recipient.

Encoding

Standard structure encoding, except that the last variable is optional.

Immediately following the request document on the reliable link is a structure which represents the unsigned partial metric data from previous hops, along with the CircuitID from the last hop. The format of this trailer is:

Structure

CircuitID	ID
MetricRequirements	Requirements

where the ID is the ID for the previous hop, and the requirement are the remaining requirements. For example, the max desired bandwidth may be lower if the previous hop could not theoretically support the original, or the required latency may be less based on latency already used.

10.4 Encryption and reliability

The encryption and reliability layers add additional headers to the circuit packets. Encryption comes before reliability. The headers look as follows.

EncryptionHeader

PreSymmetric	IV
Hash	Verification

where IV is the initialization vector, and Verification is a hash of the data that follows it, and is part of the encrypted block.

ReliabilityHeader

U32	SendSeq
U32	AckSeq
U32	WindowSize
U8	Flags

where the meaning of the parts is as described in the reliability protocol section.

Chapter 11

Document Transfer Protocol

The document transfer protocol is the mechanism used to retrieve a document based on it's name and type. The mechanism used is similar to DNS. DTP is run over a circuit, it's service name is simply, "DTP". It uses the reliability protocol over the circuit protocol to provide reliable stream service. It can be run encrypted or not, although it accepting the requested connection is the decision of the DTP server.

Over the DTP link, document requests are sent, and documents are returned. There is a redirection process, which causes the user to be pointed to another DTP server. Instead, the primary DTP server can also recursively perform additional lookups itself and return the results directly. This decision is left to the implementer, and could be based on the user requesting, the originating node, etc.

The document request looks as follows:

DocumentRequest

U8	Options (Reserved, set to 0)
DocumentType	Type
DotName	Name

and arrives over the reliable link exactly as shown above.

If the DTP server knows the value of the current document, it returns it, which simply means sending the document over the reliable link.

Otherwise, there are two special types of documents which may be used to redirect. The first is the logical name document

11.0.1 TYPE: LogicalNameBody**Description**

The logical name document redirects the document which is its key and the type described in its body to another document of the same type with a different key. It must be signed by an identity equal to or more general than its key.

Structure

DocumentType	Type
DotName	RedirectKey

where Type is the type of the document that should be redirected, and RedirectKey is the new key to redirect to.

Encoding

Standard structure encoding.

Note: Unlike most documents, there can be more than one redirection document with the same key, so long as they are redirecting different types of documents

The other, more important mechanism for redirection is the document location mechanism. The document location mechanism allows certain portions of the dot path tree to be related to certain DTP servers. If a server does not know the document, it can redirect toward a server that does. At worst, the root DTP servers know, if not the document, who is the authoritative machines for the more general portions of the sub-tree, and each of those nodes should either know the document or another more specific source, until the document is resolved.

11.0.2 TYPE: DocumentLocationBody

This is a pointer for a given portion of the dot path tree to a document server. If there are multiple records which are more general than the record being lookup up, the most specific one should be used. All DTP servers should have the location of one or more root servers. All servers authoritative for a given sub-tree should have all

documents therein, or more specific location documents for all the sub-trees of their sub-tree that they do not hold the full data for.

The key of the document location document is the sub-tree served. It must be signed by an identity equal to or more general than it's key.

The structure of the document is as follows:

Structure

DotPath

NodeName

where NodeName is a node which is authoritative for the sub-tree which is the key of the document.

Encoding

Standard structure encoding.

There can also be more than one document location document for the same key, since there can be multiple redundant, fully authoritative servers.

Chapter 12

Optional Extensions

12.1 Multicast Extension

This extension is used to allow a single node to broadcast the same stream of data to multiple nodes in an efficient manner. To allow this, the format of a circuit is modified slightly for multicast circuits (which are kept in a separate table from normal circuits). At each node, a circuit instead of connecting a single CircuitSide to another single CircuitSide, connects one "root" side to one or more "branch" sides.

In addition, there is a Channel which is unique to the sending node, and is used along with the sending node to key multicast flows for combination purposes.

12.1.1 Data Flow

When circuit data is received on the root side, it is duplicated and sent out to each of the branch sides. No actual data flows in the reverse direction (from branch to root), but there must be empty keep-alives to keep the circuit open. When a keep-alive is received via a branch, it is forwarded toward the root if there has not been a keep alive from that branch for one half the circuit timeout value, otherwise it is dropped. When a node has not received a keep-alive from a given branch for the timeout time,

it drops the branch. If there are no more branches, the entry in the multicast table is removed.

12.1.2 Circuit Establishment

Multicast circuits must be routed from the root outward. Thus if a node wishes to join the multicast it must send a multicast prerequest document (defined below) to the multicast sender requesting that it join the multicast. Once this is done, or in the case of the multicast sender initiating, the multicast sender sends a Multicast Request document, which travels outward. It is routed like a circuit request with two exceptions:

1. Upon receiving a multicast request from the same initial sender via the same side, a branch is made toward the destination, not a new circuit.
2. There may be an additional routing preference toward send data toward the same path as an existing branch.

12.1.3 TYPE: MulticastPreRequestBody

Description

Send via a non-sender to the sender to request a circuit establishment be made in an outgoing direction. It is forwarded toward the `SendingNode` and should be signed by the requesting user.

Structure

<code>DotName</code>	<code>DestinationNode</code>
<code>DotName</code>	<code>SendingNode</code>
<code>DotName</code>	<code>Channel</code>
<code>MetricRequirements</code>	<code>Requirements</code>
<code>U8</code>	<code>Flags</code>
<code>PublicEncrypted</code>	<code>CircuitKey</code>

where

`DestinationNode` is the final destination of the circuit

`SendingNode` is the node multicasting, the destination for this pre-request

`Channel` is the name of the channel desired on the other side

`Requirements` are the QOS and other metric requirements of the circuit.

`Flags` is the flags for the circuit, currently defined are bit 0 (the lowest bit) which is encryption.

`CircuitKey` is only part of the structure if the flag for encryption is set. If so, it is the symmetric encryption key, public key encrypted to the recipient.

Encoding

Standard structure encoding, except that the last variable is optional.

12.1.4 TYPE: MulticastRequestBody

Description

Sent via the sender to establish a circuit. Signed by the sending node. Followed by a PreRequest, either the one sent to initial this request, or one made up by the sender

Structure

Hash

PrerequestHash

where the PrerequestHash is the hash of the pre-request which will follow. The point of this mechanism is to allow the requesters permissions to be used for the QOS provisioning, but to make certain the sending node accepted the multicast request

Encoding

Standard structure encoding

12.1.5 Destruction

On receiving a circuit terminate request from one of the branches, the circuit terminate request is forwarded, and the circuit destroyed, if there is only one branch. If there is more than one branch, the branch in question is simply removed. A circuit terminate request from the root is forwarded to all branches, then the circuit is removed.

12.2 Automatic Splitting

Automatic splitting is a process by which a larger meta-node can split into various smaller meta-nodes. It is useful for large networks. To accomplish this mechanism a new type of meta-node is made, called a 'logical' meta-node. This differs from the standard 'administrative' meta-node, which is established by out of band key exchange. The logical meta-node gets it's authority from a number of nodes within the logical meta-node. This involves a new document, called the petition document, which establishes the logical meta-node key, and is signed by many members of the same administrative meta-node, which the logical meta-node must be a sub-part of. Additionally, these same nodes (the signers) are given new identities signed by the logical meta-node key.

This allows the creation of a logical network controller, which is a node which is promoted to the network controller status by having it's key signed by it's peers as per above. These logical network controllers can be ineffective, but not destructive. The reason is that the two purposes of the network controller: aggregate routes, and signing inter-meta-node links, both can be verified externally. In the case of route

aggregation, each node can do route aggregation themselves, and may from time to time do so to check the current logical network controller and deny further support (let their current signature timeout) of the NC in the future. The information about inter-meta-node connectivity will not be propagated unless the other side of the meta-node link cosigns, which will not happen if the results are incorrect.

In the case where nodes determine that a logical NC is behaving badly, they can create another, and so long as there is at least one functioning NC within the meta-node, all should work correctly, the malfunctioning (or malicious) NC will have no effect.

The actual process of a split involves turning an administrative or logical meta-node which is composed directly of node (lowest level) into multiple smaller meta-nodes which will now form the main meta-node (leveling), or breaking a current logical meta-node into more than one logical meta-node (dividing).

In both these cases, an NC decides that a split is desirable. In the first case, it is the higher level NC, and in the second, the NC which is sub-dividing. This involves a split-request message. This message (which is a document signed by the NC) is sent to the closest administrative NC (if the current NC is a logical NC) for a co-signature, then flooded through the meta-node, and contains information regarding the reassignment of the nodes currently in it's meta-node (which node to which sub-group), and the administrative name of the suggested NC's for each sub-group, as well as the network names. The split request specifies a time when the split is to occur. In the case of a node receiving multiple split requests, the one which is to occur the SOONEST is selected, exact identical times are disambiguated via the dictionary ordering of the sending NC's node name.

Before the split is set to occur, nodes should sign the new logical NC's petition document, as well as receive their new identity keys from the logical NC. These keys are the administratively assigned local portion of the node, followed by the new networks full name, and are signed by the NC's petition key.

The detailed process is:

1. The splitting meta-node's network control sends a split request directly to the closest administrative network controller (unless it is the nearest ANC)
2. The ANC cosigns the request and floods it.
3. The newly selected LNC's as well as non-endorsed nodes that so desire, send a 'network-controller-petition' message, containing the node's public key to all nodes involved.
4. Each node selects one or more candidates, and signs the petition
5. Upon receiving the signature, the meta-node in question signs with it's petition key the new identity of the node and returns it to the node in question
6. Upon receiving the minimum required votes, the candidate node sends it's signed petition to the nearest ANC
7. The ANC signs a LNC key for the newly appointed candidate, and send it back.
8. the new LNC floods the local network with it's signed LNC key document, it is now a logical node controller.
9. If none of a node's candidates has achieved a high enough vote count by the time the end of the election (the beginning of the split) is set to occur, the node chooses one or more of the candidates that has a high enough vote count, and signs it's petition and receives it's identity from it
10. If no candidates achieves the minimum required voted before the time specified, the split is canceled.

12.2.1 Division Choice Function

While the network control which initially decides to split can choose any method it wished to decide the organization of the split (in the sense of which nodes go to which side), there are both requirements and optimization criteria. The requirements are that each sub-group is fully internally connected, that is, each member of a sub-group can reach every other member without leaving the sub-group. Additionally, there may be a number of criteria which makes some breakdowns more desirable.

To find a good breakdown a modified method of simulated annealing is suggested. First, a fitness must be chosen which gives a value to each possible breakdown, an example might be:

$$w_1(N_a - N_b)^2 + w_2 (\text{Avg}(\text{Bandwidth}(x) : x \in \text{Inter}) / \text{Avg}(\text{Bandwidth}(x) : x \in \text{Intra}))$$

where the set *Inter* is the set of inter-group connections, and *Intra* is the set of intra-group connections, *Bandwidth(X)* is a function that measures bandwidth, w_1 and w_2 are weights, and N_a and N_b are the number of nodes in each sub-group. The goal is to minimize the value of the fitness function. This function must be normalized to a value between 0 (best) and 1 (worst).

The annealing method works as follows:

1. All nodes are initially put in sub-group A, with the exception of one randomly chosen node on the edge of the original group is put in sub-group B
2. The temperature T is initialized to 1
3. A random edge node (one which is on the edge of a sub-group) is chosen for evaluation
4. If the edge node when switched from group A to B leaves the resulting breakdown in a legal state (each group fully internally connected, at least one node in each group), the fitness of the original is evaluated, called F_{orig} as is the fitness of the network with the chosen node switched F_{new}
5. The node is switched with a probability of $0.5 + \frac{F_{orig} - F_{new}}{T}$
6. Steps 3 to 5 are repeated with continuously falling temperatures.
7. The final temperature is fixed at zero, and all edge nodes are evaluated until the system stabilizes

12.2.2 Split document details

12.2.3 TYPE: SplitRequestBody

Description

This document is sent via the NC desiring to split the network under it's control. It is signed by the NC for it's trip to the ANC if needed, and cosigned by the ANC before it is flooded.

Structure

DotPath	SplittingNode
Set<DotParh>	SuggestedNCs
Set<DotPath>	SubGroupA
Set<DotPath>	SubGroupB
TimeLocation	SplitTime
VarInteger	RequiredVotes
U8	SplitType

Encoding

Standard structure encoding

12.2.4 TYPE: ControllerPetitionBody

Description

A document which represents the candidacy of a given Node in relation to a split (which must follow this document). Should be signed by the candidate. Also sent (with the required number of votes as signatures) to the ANC once a candidate has gotten sufficient votes. Also sent signed by a single node to represent a vote cast by that node

Structure

DotPath	CandidateName
SymetricHash	HashOfSplitRequest
PublicKey	TheKey

Encoding

Standard structure encoding

12.2.5 TYPE: LogicalKeyAssignment

Description

A document sent by the a candidate, signed by their logical key, to nodes which have voted for it.

Structure

DotPath
DotPath

LogicalName
AdministrativeName

Encoding

Standard structure encoding

12.2.6 TYPE: LogicalNetworkControlAuthorization**Description**

Sent by the ANC once it is given a Controller Petition signed by the proper number of nodes

Structure

DotPath
PublicKey

CandidateName
TheKey

Encoding

Standard structure encoding

Part3

Claims

1. *Public Trust Network* A method of organization non-trusting parties with no central administration so that parties can securely share resources and communicate with an expected quality of service.
2. *Hierarchical Networking System* of organizing multiple nodes in a hierarchical relationship structure to efficiently inter communicate in a way that optimizes throughput by utilizing hierarchical pathing to provide the shortest route between two nodes. Network has the same structure on every level of the hierarchy. The internode hierarchical relationship is used.
3. *Resource Inheritance and Delegation* A system that controls resource allocation through inheritance aggregation and hierarchical rights enforcement while allowing for dynamic sub-division and delegation of resource control.
4. *Document exchange networking* A method of network communication that controls node interaction through the exchange of signed documents.
5. *Self healing QOS* A decentralized system for maintaining resource guarantees (quality of service, QOS) dynamically through path rerouting and recombination.
6. *Multicast synchronization* A broadcast communication system that allows a single source broadcast to be independently rebroadcast in multiple different time frame groupings.
7. *Network self authentication* A highly scaleable authentication method that requires the client making a request to a host to securely process portions of authentication request itself.
8. *Global root key* A method of distributing public and private keys in a way which gives each sovereign entity control over key redistribution and allows inter key cooperation when required.
9. *Cryptographic dynamic routing* A process of utilizing a set of cryptographic primitives (algorithms) to secure and control dynamic routing at the protocol level.
10. *Hierarchical dynamic routing* A process of choosing communication routes dynamically based on a hierarchical relationship of the nodes comprising a network.
11. *Non-DOS link establishment* A method of establishing a link between nodes that reduces DOS by requiring the node requesting the establishment to perform a computation that is more complex than the computation needed to perform the key exchange.
12. *Anonymous link level* A method of establishing a communications link between two nodes through a cryptographic authentication process, such that the identities of the two nodes are not externally visible over the link.
13. *Hierarchical self proving identity* A decentralized method of identity inheritance that is self-authenticating.
14. *Extensible aggregate QOS metrics* A decentralized method of quantifying aggregate resource availability for node paths in a network that optimizes routing and distributes communication load.
15. *Dynamically distributed network policy documents* A process to allow permission based network policy control via cryptographically signed

documents which are automatically distributed to the relevant nodes within the network.

16. *QOS division and delegation* A method of assigning ownership and priority to network resources which allows resources to be sub divided for use and delegation.
17. *Cryptographic hierarchical inheritance of permission (Priority)* A method of cryptographically verifying aggregated inheritance of permissions and enforcing resource allocation by aggregated inheritance priority.
18. *Mesh network subdivision (optimized clustering through simulated annealing)* A decentralized autonomous method of subdividing a network by optimized clustering; through simulated annealing.
19. *Relative QOS over non QOS link* A method of enforcing resource allocation levels on links between network nodes that utilize protocols that do not support QOS in their definition or implementation. A network bridge that enables backwards compatibility with existing network protocols that provides relative QOS.
20. *Branched circuit group broadcast* A method of broadcasting a communication between nodes of a network that redistribute the communication as needed dependent on the hierarchical relationship of the intended recipients.
21. *Decentralize Routing anti-Recurşion* Technique for preventing circular routes in decentralized dynamic routing networks.
22. *QOS aggregate dynamic routing* A method of decentralized routing that optimally chooses routes based on the current utilization of network resources and the amount of resources requested for a connection.
23. *Secure dynamic address assignment* A cryptographic method of dynamically assigning local network addresses.
24. *Document rate limiting anti-DOS* A method of protecting the communication of documents from DOS (denial of service attacks) by limiting the rate at which document communication can occur.
25. *Dynamic link QOS (flooding updates)* A process by which each node of the network is informed of the current resource availability of all links that comprise the network with as little inter-node communication as possible. This is done through full and partial hierarchical flooding techniques.
26. *Secure threshold election* A method of participant, candidate, election enforcement to allow administrative control of elections on networks comprised of non-trusting nodes.

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☐ **FADED TEXT OR DRAWING**
- ☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☐ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER:** _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.